INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Communication Efficiency in Self-stabilizing Silent Protocols

Stéphane Devismes° — Toshimitsu Masuzawa† — Sébastien Tixeuil⋆,‡
° VERIMAG, Université Joseph Fourier, Grenoble I (France)
† Graduate School of Information Science and Technology, Osaka University, Osaka (Japan)
⋆ Université Pierre et Marie Curie - Paris 6, LIP6, France

‡ INRIA project-team Grand Large

N° 6731

November 2008

Thème NUM

ISSN 0249-6399 ISRN INRIA/RR--6731--FR+ENG

Communication Efficiency in Self-stabilizing Silent Protocols

Stéphane Devismes° , Toshimitsu Masuzawa† , Sébastien Tixeuil⋆,‡
° VERIMAG, Université Joseph Fourier, Grenoble I (France)
† Graduate School of Information Science and Technology, Osaka University, Osaka (Japan)
⋆ Université Pierre et Marie Curie - Paris 6, LIP6, France
‡ INRIA project-team Grand Large

Thème NUM — Systèmes numériques
Projet Grand Large

Rapport de recherche n° 6731 — November 2008 — 30 pages

Abstract: Self-stabilization is a general paradigm to provide forward recovery capabilities to distributed systems and networks. Intuitively, a protocol is self-stabilizing if it is able to recover without external intervention from any catastrophic transient failure.

In this paper, our focus is to lower the communication complexity of self-stabilizing protocols *below* the need of checking every neighbor forever. In more details, the contribution of the paper is threefold: *(i)* We provide new complexity measures for communication efficiency of self-stabilizing protocols, especially in the stabilized phase or when there are no faults, *(ii)* On the negative side, we show that for non-trivial problems such as coloring, maximal matching, and maximal independent set, it is impossible to get (deterministic or probabilistic) self-stabilizing solutions where *every* participant communicates with less than every neighbor in the stabilized phase, and *(iii)* On the positive side, we present protocols for coloring, maximal matching, and maximal independent set such that a fraction of the participants communicates with *exactly one* neighbor in the stabilized phase.

Key-words: self-stabilization, lower bounds, communication complexity, coloring, maximal matching, maximal independent set

Unité de recherche INRIA Futurs
Parc Club Orsay Université, ZAC des Vignes,
4, rue Jacques Monod, 91893 ORSAY Cedex (France)
Téléphone : +33 1 72 92 59 00 — Télécopie : +33 1 60 19 66 08

Efficacité des Communications des les Protocoles Auto-stabilisants Silencieux

Résumé : L'auto-stabilisation est un paradigme général pour assurer la reprise sur erreur dans les réseaux et les systèmes distribués. Un algorithme réparti est auto-stabilisant si, après que des fautes et des attaques aient frappé le système et l'aient placé dans un état quelconque, le système corrige cette situation catastrophique sans intervention extérieure en temps fini.

Dans cet article, nous nous concentrons sur la complexité des communications des algorithmes auto-stabilisants au delà du besoin de vérifier infiment souvent tous les voisins de chaque processus. La contribution de l'article peut être résumée en trois points principaux: *(i)* nous proposons de nouvelles mesures de complexité pour les communications des protocoles auto-stabilisants, e particulier dans la phase stabilisée ou quand il n'y a pas de fautes ; *(ii)* nous montrons que pour des problèmes non-triviaux tels que le coloriage, le mariage maximal, et l'ensemble maximal indépendant, il est impossible d'obtenir des solutions (déterministes ou probabilistes) auto-stabilisantes ou *chaque* participant communique avec moins que tous ses voisins dans la phase stabilisée ; *(iii)* nous présentons des protocoles pour le coloriage, le mariage maximal, et l'ensemble maximal indépendant tels qu'une fraction des participants communique avec *exactement un* voisin lors de la phase stabilisée.

Mots-clés : auto-stabilisation, bornes inférieures, complexité des communications, coloriage, mariage maximal, ensemble maximal indépendant

1 Introduction

Self-stabilization [8] is a general paradigm to provide forward recovery capabilities to distributed systems and networks. Intuitively, a protocol is self-stabilizing if it is able to recover without external intervention from any catastrophic transient failure. Among the many self-stabilizing solutions available today [9], the most useful ones for real networks are those that admit efficient implementations.

Most of the literature is dedicated to improving efficiency after failures occur, *i.e.*, minimizing the stabilization time - the maximum amount of time one has to wait before failure recovery. While this metric is meaningful to evaluate the efficiency in the presence of failures, it fails at capturing the overhead of self-stabilization when there are no faults, or after stabilization. In order to take forward recovery actions in case of failures, a self-stabilizing protocol has to gather information from other nodes in order to detect inconsistencies. Of course, a *global* communication mechanism will lead to a large coverage of anomaly detection [15] at the expense of an extremely expensive solution when there are no faults, since information about every participant has to be repetitively sent to every other *participant*. As pointed out in [5], the amount of information that has to be gatherered highly depends on the task to be solved if only the output of the protocol is to be used for such anomaly detection. The paper also points out that more efficient schemes could be available for some particular implementations. However, to the best of our knowledge, the minimal amount of communicated information in self-stabilizing systems is still *fully local* [3, 4, 5]: when there are no faults, every participant has to communicate with every other *neighbor* repetitively.

In this paper, our focus is to lower the communication complexity of self-stabilizing protocols *below* the need of checking every neighbor. A quick observation shows that non-existent communication is impossible in the context of self-stabilization: the initial configuration of the network could be such that the specification is violated while no participant is sending nor getting neighboring information, resulting in a deadlock. On the other side, there exist problems (such as coloring, maximal matching, maximal independent set) that admit solutions where participants only have to communicate with their full set of neighbors. We investigate the possibility of intermediate solutions (*i.e.* where participants communicate repetitively only with a *strict subset* of their neighbors) that would lead to more efficient implementations in stabilized phase or when there are no faults. Good candidates for admitting such interesting complexity solutions are *silent* protocols [10]: a silent protocol is a self-stabilizing protocol that exhibits the additionnal property that after stabilization, communication is fixed between neighbors (that is, neighbors repetitively commu-

nicate the same information for every neighbor forever). We thus concentrate on lowering communication complexity requirements for silent self-stabilizing protocols.

In more details, the contribution of the paper is threefold:

1. We provide new complexity measures for communication efficiency of self-stabilizing protocols, especially in the stabilized phase or when there are no faults. Our notion of communication efficiency differs from the one introduced in [16] (that was subsequently used for fault-tolerant non-self-stabilizing systems [16, 1, 2] and then extented to fault-tolerant self-stabilizing systems – *a.k.a. ftss* – [7]). The essential difference is that the efficiency criterium of [16] is *global* (eventually only $n-1$ communication channels are used) while our notion is local (eventually processes only commnunicate with a strict subset of their neighbors). As noted in [16, 1, 2, 7], global communication efficiency often leads to solutions where one process needs to periodically send messages to every other process. In contrast, with our notion, the communication load is entirely distributed and balanced.

2. On the negative side, we show two impossibility results holding for a wide class of problems. This class includes many classical distributed problems, *e.g.*, coloring [12], maximal matching [17], and maximal independent set [13]. We first show that there is no (deterministic or probabilistic) self-stabilizing solutions for such problems in arbitrary anonymous network where *every* participant has communicate with a strict subset of its neighbors once the system is stabilized. We then show that it is even more difficult to self-stabilize these problems if the communication constraint must always hold. Indeed, even with symmetry-breaking mechanisms such as a leader or acyclic orientation of the network, those tasks remain impossible to solve.

3. On the positive side, we present protocols for coloring, maximal matching, and maximal independent set such that a fraction of the participants communicates with *exactly one* neighbor in the stabilized phase.

The remaining of the paper is organized as follows. Section 2 presents the computational model we use throughout the paper. We introduce in Section 3 new complexity measures for communication efficiency of self-stabilizing protocols. Sections 4 and 5 describe our negative and positive results, respectively. Section 6 provides some concluding remarks and open questions.

2 Model

A *distributed system* is a set Π of n communicating state machines called *processes*. Each process p can directly communicate using *bidirectional* media with a restricted subset of processes called *neighbors*. We denote by $\Gamma.p$ the set of p's neighbors and by $\delta.p$ the degree of p, *i.e.*, the size of $\Gamma.p$. We consider here distributed systems having an *arbitrary connected topology*, modelized by an undirected connected graph $G = (\Pi, E)$ where E is a set of m edges representing the bidirectional media between neighboring processes. In the sequel, Δ denotes the degree of G and D its diameter.

We assume that each process p can distinguish any two neighbors using *local indices*, that are numbered from 1 to $\delta.p$. In the following, we will indifferently use the *label* q to designate the process q or the local index of q in the code of some process p. We will often use the *anonymous* assumption which states that the processes may only differ by their degrees.

Communications are carried out using a finite number of *communication variables* that are maintained by each process. Communication variables maintained by process p can be read and written by p, but only read by p's neighbors. Each process p also maintains a finite set of *internal variables* that may only be accessed by p. Each variable ranges over a fixed domain of values. We use uppercase letters to denote communication variables and lowercase ones to denote internal variables. Some variables can be *constant*, that is, they have a determined fixed value. In the following, we will refer to a variable v of the process p as $v.p$. The *state* of a process is defined by the values of its (communication and internal) variables. A *configuration* is an instance of the states of all processes. The *communication state* of a process is its state restricted to its communication variables. A *communication configuration* is an instance of the communication states of all processes.

A *protocol* is a collection of n sequential *local algorithms*, each process executing one local algorithm. A process updates its state by executing its local algorithm. A local algorithm consists of a finite set of guarded actions of the form $\langle guard \rangle \rightarrow \langle action \rangle$. A guard is a Boolean predicate over the (communication and internal) variables of the process and the communication variables of its neighbors. An *action* is a sequence of statements assigning new values to its (communication and internal) variables. An action can be executed only if its guard is *true*. We assume that the execution of any action is *atomic*. An action is said *enabled* in some configuration if its guard is *true* in this configuration. By extention, we say that a process is enabled if at least one of its actions is enabled.

A *computation* is an infinite sequence $(\gamma_0 s_0 \gamma_1), (\gamma_1 s_1 \gamma_2), \ldots (\gamma_i s_i \gamma_{i+1}), \ldots$ such that for any $i \geq 0$: *(i)* γ_i is a configuration, *(ii)* s_i is a non-empty subset of processes

chosen according to a *scheduler* (defined below), and *(iii)* each configuration γ_{i+1} is obtained from γ_i after all processes in s_i execute from γ_{i-1} one of their enabled actions, if any.[1] Any triplet $(\gamma_i s_i \gamma_{i+1})$ is called a *step*. Any finite sequence of consecutive steps of C starting from γ_0 is a *prefix* of C. A *suffix* of C is any computation obtained by removing a finite sequence $(\gamma_0 s_0 \gamma_1), \ldots, (\gamma_k s_k \gamma_{k+1})$ from C. The suffix associated to the prefix $(\gamma_0 s_0 \gamma_1) \ldots, (\gamma_{i-1} s_{i-1} \gamma_i)$ is the suffix of C starting from γ_i. A configuration γ' is said *reachable* from the configuration γ if and only if there exists a computation starting from γ that contains the configuration γ'.

A *scheduler* is a predicate on computations that determines which are the possible computations. In this paper, we assume a *distributed fair scheduler*. *Distributed* means that any non-empty subset of processes can be chosen in each step to execute an action. *Fair* means that every process is selected infinitely many times to execute an action. We assume priority on the guarded actions that are induced by the order of appearance of the actions in the code of the protocols. Actions appearing first have higher priority than those appearing last.

To compute the time complexity, we use the notion of *round* [11]. This notion captures the execution rate of the slowest process in any computation. The first *round* of an computation C, noted C', is the minimal prefix of C where every process has been activated by the scheduler. Let C'' be the suffix associated to C'. The second *round* of C is the first round of C'', and so on.

2.1 Self-stabilization

We now formally define the notions of *deterministic self-stabilization* [8] (simply referred to as self-stabilization) and *probabilistic self-stabilization* [14].

A configuration *conforms* to a predicate if this predicate is satisfied in this configuration; otherwise the configuration *violates* the predicate. By this definition every configuration conforms to the predicate *true* and none conforms to the predicate *false*. Let R and S be predicates on configurations of the protocol. Predicate R is *closed* with respect to the protocol actions if every configuration of any computation that starts in a configuration conforming to R also conforms to R. Predicate R *converges* to S if R and S are closed and every computation starting from a configuration conforming to R contains a configuration conforming to S.

Definition 1 (Deterministic Self-Stabilization) *A protocol* deterministically stabilizes *to a predicate R if and only if* true *converges to R.*

[1] If all processes in s_i are disabled in γ_i, then $\gamma_{i+1} = \gamma_i$

Definition 2 (Probabilistic Self-Stabilization) *A protocol* probabilistically stabilizes *to a predicate R if and only if* true *converges to R with probability 1.*

In any protocol that stabilizes to the predicate R, any configuration that conforms to R is said *legitimate*. Conversely, any configuration that violates R is said *illegitimate*.

2.2 Silence

All protocols presented in this paper are *silent*. The notion of *silent protocol* has been defined in [10] as follows:

Definition 3 (Silent Protocol) *An protocol is* silent *if and only if starting from any configuration, it converges to a configuration after which the values of its communication variables are fixed.*

In the remaining of the paper, we will call *silent configuration* any configuration from which the values of all communication variables are fixed.

3 New Measures for Communication Efficiency

3.1 Communication Efficiency

In this paper, we are interested in designing self-stabilizing protocols where processes do not communicate with all their neighbors during each step. The `k`-*efficiency* defined below allows to compare protocols following this criterium.

Definition 4 (k-efficient) *A protocol is said to be* `k`*-efficient if in every step of its possible computations, every process reads communication variables of at most* `k` *neighbors.*

Note that in this paper, we only present 1-*efficient* protocol. Note also that every distributed self-stabilizing protocol is trivially Δ-*efficient*.

3.2 Space complexity

To be able to compare the space complexity of distributed algorihtms, we distinguish two complexity criteria.

Definition 5 (Communication Complexity) *The* communication complexity *of a process p is the maximal amount of memory p reads from its neighbors in any given step.*

Example: In our coloring protocol (Figure 7, page 16), in any step a process only reads the color ($\Delta + 1$ states) of a single neighbor, so in this protocol the communication complexity is $\log(\Delta + 1)$ bits per process. By contrast, a traditional coloring protocol that reads the state of every neighbor at each step has communication complexity $\Delta \log(\Delta + 1)$.

Definition 6 (Space complexity) *The* space complexity *of a process p is the sum of the local memory space (that is, the space needed for communication and internal variables) and the communication complexity of p.*

Example: In our coloring protocol (Figure 7, page 16), the communication complexity is $\log(\Delta + 1)$ bits per process and the local memory space of any process p is $\log(\Delta + 1) + \log(\delta.p)$ bits ($\log(\Delta + 1)$ for the C-variable and $\log(\delta.p)$ for the *cur*-variable). So a process p has a space complexity of $2\log(\Delta + 1) + \log(\delta.p)$ bits.

3.3 Communication Stability

In our protocols, some processes may read the communication variables of every neighbor forever, while other processes may eventually read the communication variable of a single neighbor. We emphasize this behavior by introducing the k-*stability* and two weakened forms: the $\Diamond$-k-*stability* and the $\Diamond$-(x, k)-*stability*.

Let $C = (\gamma_0 s_0 \gamma_1), \ldots (\gamma_{i-1} s_{i-1} \gamma_i), \ldots$ be a computation. Let $R_p^i(C)$ be the set of neighbors from which p reads some communication variables in step $(\gamma_i s_i \gamma_{i+1})$. Let $R_p(C) = |R_p^0(C) \cup \ldots \cup R_p^i(C) \cup \ldots|$.

Definition 7 (k-Stable) *A protocol is* k-stable *if in every computation C, every process p satisfies $R_p(C) \leq k$.*

Observe that every protocol is Δ-*stable*. Note also that any k-*stable* protocol is also k-*efficient* but k-*efficient* protocols are not necessarily k-*stable*.

Definition 8 ($\Diamond$-k-Stable) *A protocol is* $\Diamond$-k-stable *if in every computation C, there is a suffix C' such that every process p satisfies $R_p(C') \leq k$.*

Definition 9 ($\Diamond$-(x,k)-Stable) *A protocol is* $\Diamond$-(x, k)-stable *if in every computation C, there are a subset $\mathcal{S}$ of x processes and a suffix C' such that every process $p \in \mathcal{S}$ satisfies $R_p(C') \leq k$.*

Similary to $\Diamond$-(x, k)-*stable*, one could define the notion of (x, k)-*stable*. However, we do not consider such a property here. Note also that the notions of $\Diamond$-k-*stable* and $\Diamond$-(n, k)-*stable* are equivalent.

4 Impossibility Results

We now provide a general condition on the output of communication variables that prevents the existence of some communication stable solutions. Informally, if the communication variables of two neighboring processes p and q can be in two states α_p and α_q that are legitimate separately but not simultaneously, there exists no $\Diamond$-k-stable solution for $k < \Delta$. This condition, that we refer to by the notion of *neighbor-completeness* is actually satisfied by every silent self-stabilizing solution to the problems we consider in the paper: vertex coloring, maximal independent set, maximal matching.

Definition 10 (neighbor-completeness) *An protocol A is said* neighbor-complete *for predicate P if and only if A is silent, self-stabilizes to P, and for every process p, there exists a communication state of p, say α_p, such that:*

1. *There exists a silent configuration where the communication state of p is α_p.*
2. *For every neighbor of p, say q, there exists a communication state of q, say α_q, such that:*
   (a) *Every configuration where the communication state of p is α_p and the communication state of q is α_q violates P.*
   (b) *There exists a silent configuration where the communication state of q is α_q.*

Theorem 1 *There is no* $\Diamond$-$\mathtt{k}$-stable *(even probabilistic)* neighbor-complete *protocol working in arbitrary anonymous networks of degree* $\Delta > \mathtt{k}$.

Proof. Assume, by the contradiction, that there exists an $\Diamond$-$\mathtt{k}$-*stable* protocol that is (deterministically or probabilistically) *neighbor-complete* for a predicate P in any anonymous network of degree $\Delta > \mathtt{k}$.

To show the contradiction, we prove that for any $\Delta > 0$, there exist topologies of degree Δ for which there is no $\Diamond$-$\mathtt{k}$-*stable* protocol that is *neighbor-complete* for P with $\mathtt{k} = \Delta - 1$. This result implies the contradiction for any $\mathtt{k} < \Delta$.

We first consider the case $\Delta = 2$. (The case $\Delta = 1$ can be easily deduce using a network of two processes and following the same construction as the one for $\Delta = 2$.) We will then explain how to generalize the case $\Delta = 2$ for any $\Delta \geq 2$.

Case $\Delta = 2$ and $\mathtt{k} = \Delta - 1$: Consider an anonymous chain of five processes p_1, p_2, p_3, p_4, and p_5. (In the following, Figure 1 may help the reader.)

By Definition, there exists a communication state of p_3, say α_3 such that:

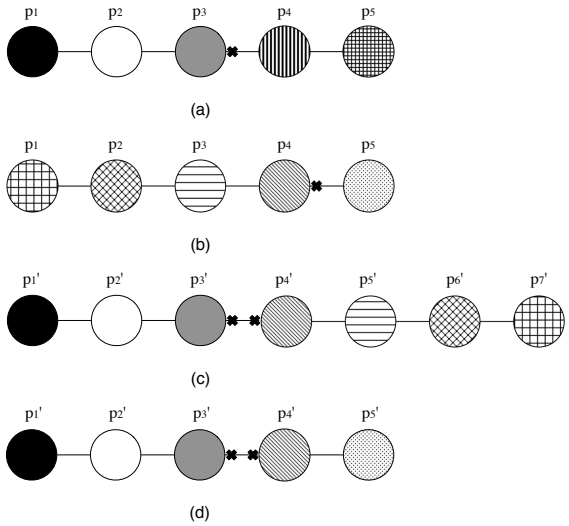

Figure 1: Construction of illegitimate silent configurations in Theorem 1 (The black crosses indicate that the communication variable of a process is not read by a neighbor)

1. There exists a silent configuration γ_3 where the communication state of p_3 is α_3.
2. For every neighbor of p_3, p_i ($i \in \{2,4\}$), there exists a communication state of p_i, say α_i, such that:
   (a) Any configuration where the communication state of p_3 is α_3 and the communication state of p_i is α_i violates P.
   (b) There exists a silent configuration γ_i where the communication state of p_i is α_i.

From the configuration γ_3, the system eventually reaches a silent configuration γ'_3 from which p_3 stops to read the communication variables of one neighbor because the degree of p_3 is equal to Δ. Without loss of generality, assume that this neighbor is p_4. (As in the configuration (a) in Figure 1.) As γ_3 is silent, the communication state of p in γ'_3 is the same as in γ_3: α_3.

Similary, from γ_4 (γ_i with $i = 4$), the system eventually reaches a silent configuration γ'_4 from which p_4 stops to read the communication variables of one neighbor $p_j \in \{3,5\}$ and where the communication state of p_4 is α_4.

Consider the two following cases:

- $p_j = p_5$. (As in the configuration (b) in Figure 1). Consider a new network of seven processes: $p'_1, \ldots, p'_7$. Assume the following initial configuration γ: Any process p'_i with $i \in \{1, 2, 3\}$ has the same state as p_i in γ'_3, p'_4 has the state of p_4 in γ'_4, p'_5 has the state of p_3 in γ'_4, p'_6 has the state of p_2 in γ'_4, and p'_7 has the state of p_1 in γ'_4. (This configuration corresponds to the configuration (c) of Figure 1.) We can then remark that p'_3 is in the same situation that p_3 in the configuration γ'_3, so p'_3 does not read the communication variables of p'_4. Similary, p'_4 does not read the communication variables of p'_3. Moreover, no process modifies the content of its communication variable, otherwise they can do the same in γ'_3 or γ'_4 and this contradicts the fact that γ'_3 and γ'_4 are silent. Hence, γ is silent and, as p'_3 and p'_4 have the same communication state in γ as p_3 in γ_3 and p_4 in γ_4, γ violates P. Thus, any computation starting from γ never converges to a configuration satisfying P, *i.e.*, protocol A is not self-stabilizing for P, a contradiction.
- $p_j = p_3$. This case is similar to the previous one: By constructing a configuration such as the configuration (d) in Figure 1, we also obtain a contradiction.

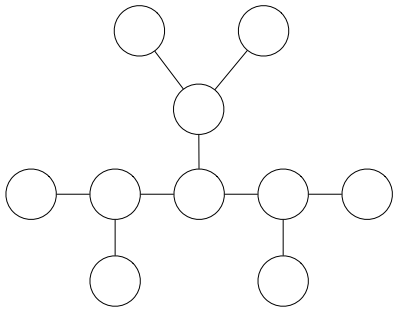

Figure 2: Generalization for $\Delta = 3$, Theorem 1

The previous proof can be generalizated for $\mathtt{k} = \Delta - 1$ and $\Delta > 2$ using a graph of $\Delta^2 + 1$ nodes where there is a node of degree Δ (the role of this node is the same as node p_3 in the case $\mathtt{k} = \Delta - 1$ and $\Delta = 2$) that is linked to Δ nodes of degree Δ. Each of these last Δ nodes being linked to $\Delta - 1$ pendent nodes. Figure 2 depicts the generalization for $\Delta = 3$.

□

In the next section, we will show that assuming a colored arbitrary network, it is possible to design $\Diamond$-$(x,1)$-*stable neighbor-complete* protocols. Actually, we use the local coloring because it allows to deduce a *dag-orientation* in the network (defined below). Theorem 2 shows that even assuming a *rooted* and/or *dag-oriented* network, it is impossible to design $\mathtt{k}$-*stable neighbor-complete* protocols for $\mathtt{k} < \Delta$.

Definition 11 (Dag-orientation) *Let $\mathcal{S}.p$ be the set of possible states of process p. We say that a system is* dag-oriented iff *for every process p, there exists a function $f_p : \mathcal{S}.p \mapsto 2^{\Gamma.p}$ and a subset $Succ.p \subseteq \Gamma.p$ such that:*

- $\forall \alpha_p \in \mathcal{S}.p$, $f_p(\alpha_p) = Succ.p$, *and*
- *The directed subgraph $G' = (\Pi, E')$ where $E' = \{(p,q), p \in \Pi \wedge q \in Succ.p\}$ is a* dag.[2]

Theorem 2 *Let $\mathtt{k} < \Delta$. There is no* $\mathtt{k}$-stable *(even probabilistic)* neighbor-complete *protocol in any arbitrary rooted and dag-oriented network.*

Proof. Assume, by the contradiction, that there exists a $\mathtt{k}$-*stable* protocol that is (deterministically or probabilistically) *neighbor-complete* for a predicate P in any arbitrary rooted and dag-oriented network.

To show the contradiction, we prove that for any $\Delta > 0$, there exists rooted and dag-oriented topologies of degree Δ where there is no $\mathtt{k}$-*stable* protocol that is *neighbor-complete* for P with $\mathtt{k} = \Delta - 1$. This result implies the contradiction for any $\mathtt{k} < \Delta$.

To that goal, we first consider the case $\Delta = 2$ (the case $\Delta = 1$ is trivial because, in this case, $\mathtt{k} = 0$ and in any $\mathtt{0}$-*stable* protocol, no process can communicate with each other). We will then explain how to generalize the case $\Delta = 2$ for any $\Delta \geq 2$.

Case $\Delta = 2$ and $\mathtt{k} = \Delta - 1$: Consider the rooted dag-oriented network presented in Figure 3. In this figure, the dag-orientation is given by the arrows and the root is the process represented as a bold circle (that is, process p_1).

Let us consider process p_2. By Definition, there exists a communication state of p_2, say α_2 such that:

1. There exists a silent configuration γ_2 where the communication state of p_2 is α_2.
2. For every neighbor of p_2, say p_i ($i \in \{1,5\}$), there exists a communication state of p_i, say α_i, such that:

[2] Directed Acyclic Graph

(a) Any configuration where the communication state of p_2 is α_2 and the communication state of p_i is α_i violates P.

(b) There exists a silent configuration γ_i where the communication state of p_i is α_i.

Consider the silent configuration γ_2. The degree of process p_2 is Δ so, from γ_2, p_2 never reads the communication variable of one of its neighbor: p_1 or p_5. So, let us consider these two cases:

1. *From γ_2, p_2 does not read the communication variables of p_5.* Consider the process p_6 in γ_2. By definition, since p_6 reads the communication variables of one of its neighbors, p_6 cannot read the one of the other neighbor forever. So, p_6 decides which process it never read only using its state in γ_2. Moreover, it cannot use the orientation to take its decision because the orientation is the same of each of its two neighbors. Depending only on its state, p_6 will decide to read the communication variable of its neighbor having the channel number $i \in \{1, 2\}$. Now there exists a possible network where p_4 is the neighbor i in the local order of p_6. Hence, we can have a configuration γ_2 similar to the configuration (a) of Figure 4: a silent configuration from which p_2 never reads the communication variables of p_5 and p_6 never reads the communication variables of p_4.

   Using a similar reasonning, we can have a silent configuration γ_5 from which p_5 never reads the communication variables of p_2 and p_4 never reads the communication variables of p_6: configuration (b) of Figure 4.

   Consider now the configuration (c) of Figure 4. In this configuration: (1) p_1, p_2, p_3, and p_6 have the same states, the same channel labelling, and so the same local views of their neighbors as in γ_2, and (2) p_4 and p_5 have the same states, the same channel labelling, and so the same local views of their neighbors as in γ_5. So, configuration (c) is silent (otherwise this means that γ_2 or γ_5 are not silent). But, as the communication states of p_2 and p_5 are respectively α_2 and α_5, configuration (c) violates P. Hence, any computation starting from configuration (c) never converges to a legitimate configuration, *i.e.*, A is not self-stabilizing for P, a contradiction.

2. *From γ_2, p_2 does not read the communication variables of p_1.* Similary to the previous case, we can also obtain a contradiction by constructing an illegitimate silent configuration such as the one shown in Figure 5 (c).

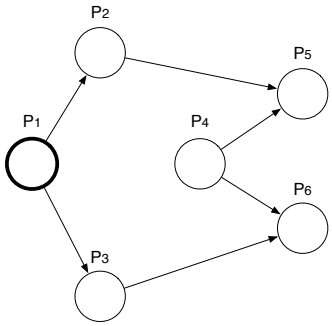

Figure 3: Network considered in Theorem 2

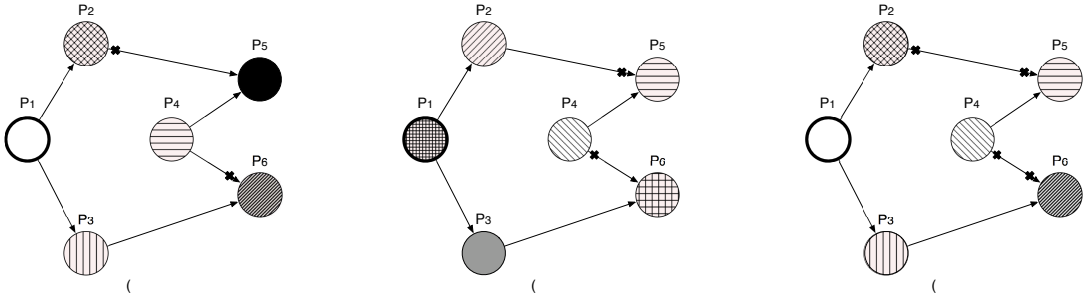

Figure 4: First case of Theorem 2

The previous proof can be generalized for $\mathtt{k} = \Delta - 1$ and $\Delta > 2$ by considering a topology where $\Delta - 2$ pendent nodes are added to each process of the network in Figure 3. (*n.b*, the arrowes must be oriented in such way that (1) p_1 and p_4 remain *sources* and (2) p_5 and p_6 remain *sink*.) Figure 6 depicts the generalization for $\Delta = 3$. □

5 Protocols

5.1 Vertex Coloring

In the *vertex coloring* problem, each process p computes a local function *color.p*. The function *color.p* outputs a value called *color* of p. The *vertex coloring predicate* is true if and only if for every process p and every p's neighbor q, $color.p \neq color.q$. In

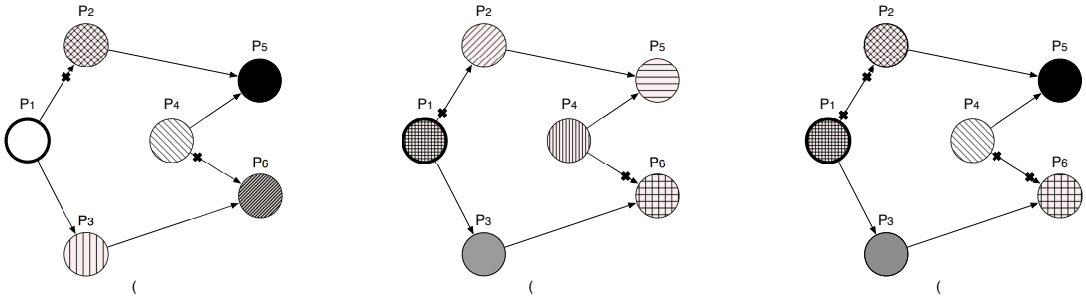

Figure 5: Second case of Theorem 2

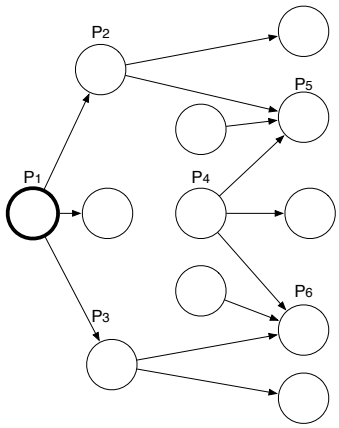

Figure 6: Generalization for $\Delta = 3$, Theorem 2

any self-stabilizing vertex coloring protocol, the legitimate configurations are those satisfying the *vertex coloring predicate*.

Communication Variable:
$C.p \in \{1 \dots \Delta + 1\}$

Internal Variable:
$cur.p \in [1 \dots \delta.p]$

Actions:

$(C.p = C.(cur.p))$	$\rightarrow$	$C.p \leftarrow random(\{1 \dots \Delta + 1\})$; $cur.p \leftarrow (cur.p \bmod \delta.p) + 1$
$(C.p \neq C.(cur.p))$	$\rightarrow$	$cur.p \leftarrow (cur.p \bmod \delta.p) + 1$

Figure 7: Protocol $\mathcal{COLORING}$ for any process p

We propose in Figure 7 a 1-*efficient* protocol that stabilizes to the *vertex coloring predicate* with probability 1. In the following, we refer to this protocol as Protocol $\mathcal{COLORING}$. Protocol $\mathcal{COLORING}$ is designed for arbitrary anonymous networks. In Protocol $\mathcal{COLORING}$, the function *color.p* just consists in outputting the value of the communication variable $C.p$. This variable takes a value in $\{1 \dots \Delta + 1\}$, $\Delta + 1$ being the minimal number of colors required to solve the problem in any arbitrary network (indeed, the protocol must operate even if the network contains a Δ-clique). The legitimate configurations of Protocol $\mathcal{COLORING}$ (*w.r.t.* the *vertex coloring predicate*) are the thoses satisfying: for every process p and every p's neighbor q, $C.p \neq C.q$.

In Protocol $\mathcal{COLORING}$, each process checks one by one the color of its neighbors in a round robin manner. The current checked neighbor is the neighbor pointed out by the internal variable *cur*. If a process detects that its color is identical to the one of the neighbor it is checking, then it chooses a new color by random in the set $\{1 \dots \Delta + 1\}$. Below, we show the correctness of Protocol $\mathcal{COLORING}$.

The following lemma is trivial because a process can change its color only if it has the same color as the neighbor it checks.

Lemma 1 *The* vertex coloring predicate *is closed in any computation of* $\mathcal{COLORING}$.

Lemma 2 *Starting from an arbitrary configuration, any computation of* $\mathcal{COLORING}$ *reaches a configuration satisfying the* vertex coloring predicate *with probability 1.*

Proof. Let $Conflit(\gamma)$ be the number of processes p having a neighbor q such that $C.p = C.q$ in γ.

Assume, by the contradiction, that there exists a configuration from which the probability to reach a legitimate configuration (*i.e.*, a configuration satisfying the *vertex coloring predicate*) is strictly less than 1.

The number of configurations is finite because any variable in the code of $\mathcal{COLORING}$ ranges over a fix domain and the number of processes (n) is finite. So, there exists a subset of illegitimate configurations S satisfying: $\forall \gamma \in S$, the probability to reach from γ a configuration γ' such that $Conflit(\gamma') < Conflit(\gamma)$ is 0.

Consider a configuration γ in S. From γ, every process executes actions infinitely often because the scheduler is fair and every process is always enabled. Let P_γ be the subset of processes p such that $\exists q \in \Gamma.p$, $C.p = C.q$ in γ. Some processes of P_γ eventually choose a new color by random in $\{1 \dots \Delta + 1\}$. Let $P^1_\gamma \subseteq P_\gamma$ be the processes that are chosen first by the scheduler to change their color. The probability to reach a configuration γ' such that $Conflit(\gamma') < Conflit(\gamma)$ when the processes of P^1_γ change their color is strictly positive because there is at least one combination of new colors such that $\forall p \in P^1_\gamma$, $\forall q \in \Gamma.p$, $C.p \neq C.q$ in γ'. Hence, the probability to reach from γ a configuration γ' such that $Conflit(\gamma') < Conflit(\gamma)$ is different of 0, a contradiction. □

By Lemmas 1, 2, Protocol $\mathcal{COLORING}$ converges from any configuration to the *vertex coloring predicate* with probability 1. Moreover, $\mathcal{COLORING}$ is 1-*efficient* because when a process p executes an action in $\mathcal{COLORING}$, it only reads the color of its neighbor pointed out by *cur.p*. Hence, follows:

Theorem 3 *Protocol $\mathcal{COLORING}$ (Figure 7) is a* 1*-efficient protocol that stabilizes to the* vertex coloring predicate *with probability 1 in any anonymous network.*

5.2 Maximal Independent Set

We now consider the *maximal independent set* (MIS) problem. An *independent set* of the network is a subset of processes such that no two distinct processes of this set are neighbors. An independent set S is said *maximal* if no proper superset of S is an independent set.

In the *maximal independent set* problem, each process p computes a local Boolean function $inMIS.p$ that decides if p is in the maximal independent set. The *MIS predicate* is $true$ if and only if the subset $\{q \in \Pi, inMIS.q\}$ is a maximal independent set of the network. In any self-stabilizing MIS protocol, the legitimate configurations are those satisfying the *MIS predicate*.

We propose in Figure 8 a 1-*efficient* protocol that stabilizes to the *MIS predicate*. In the following, we refer to this protocol as Protocol $\mathcal{MIS}$. In [13], authors propose a self-stabilizing MIS protocol working in arbitrary networks assuming that processes have *global* identifiers that can be ordered. Here, the proposed protocol works in arbitrary networks too, but assuming that (1) each process p holds a *local* identifier

Communication Variable:
$S.p \in \{Dominator, dominated\}$

Communication Constant:
$C.p$: color

Internal Variable:
$cur.p \in [1 \dots \delta.p]$

Actions:

$(S.(cur.p) = Dominator \wedge C.(cur.p) \prec C.p \wedge S.p = Dominator)$	$\rightarrow$	$S.p \leftarrow dominated$
$[(S.(cur.p) = dominated \vee C.p \prec C.(cur.p)) \wedge (S.p = dominated)]$	$\rightarrow$	$S.p \leftarrow Dominator$; $cur.p \leftarrow (cur.p \bmod \delta.p) + 1$
$(S.p = Dominator)$	$\rightarrow$	$cur.p \leftarrow (cur.p \bmod \delta.p) + 1$

Figure 8: Protocol $\mathcal{MIS}$ for any process p

$C.p$ (*i.e.*, a "color" that is unique in the neighbourhood) and (2) the local identifiers are ordered following the relation $\prec$. Using such colors is very usefull because it gives a DAG-orientation of the network, as shown below:

Theorem 4 *Let E' be the set of oriented edges such that $(p, q) \in E'$ if and only if p and q are* neighbors *and $C.p \prec C.q$. The oriented graph $G' = (\Pi, E')$ is a directed acyclic graph (*dag*).*

Proof. Assume, by the contradiction, that there is a cycle $p_0 \dots p_k$ in G'. Then, there is an oriented edge (p_k, p_0) which means that: (1) p_0 and p_k are neighbors and (2) $C.p_k \prec C.p_0$. Now, $\forall i \in [0 \dots k-1]$, $C.p_i \prec C.p_{i+1}$. So, by transitivity, $C.p_0 \prec C.p_k$ and this contradicts (2). □

In Protocol $\mathcal{MIS}$, any process p maintains the communication variable $S.p$ that has two possible states: *Dominator* or *dominated*. $S.p$ states if p is in the independent set (*Dominator*) or not (*dominated*). Hence, in Protocol $\mathcal{MIS}$, the function $inMIS.p$ just consists in testing if $S.p = Dominator$. The legitimate configurations of Protocol $\mathcal{MIS}$ are those satisfying:

1. $\forall p \in \Pi$, $(S.p = Dominator) \Rightarrow (\forall q \in \Gamma.p, S.q = dominated)$.
2. $\forall p \in \Pi$, $(S.p = dominated) \Rightarrow (\exists q \in \Gamma.p, S.q = Dominator)$.

The first condition states that the set of *Dominators* is an independent set, while the second condition states that the independent set is maximal.

We now outline the principles of Protocol $\mathcal{MIS}$. First, we use the internal variable, *cur*, to get the communication efficiency: a process p can only read the communication state of the neighbor pointed out by $cur.p$. Then, depending of $S.p$, each process p adopts the following strategy:

- If $S.p = Dominator$, then p checks one by one (in a round robin manner) the communication states of its neighbors until it points out a neighbor q that is also a *Dominator*. In such a case, either p or q must become *dominated* to satisfy Condition 1. We then use the colors to make a deterministic choice between p and q. That is, the one having the greatest color (*w.r.t.* $\prec$) becomes *dominated*. Note that in a legitimate configuration, every *Dominator* process continues to check its neighbors all the time.

- If $S.p = dominated$, then p must have the guarantee that one of its neighbor is a *Dominator*. Hence, p switches $S.p$ from *dominated* to *Dominator* if the neighbor it points out with $cur.p$ is not a *Dominator* (*i.e.*, $S.(cur.p) = dominated$). Also, to have a faster convergence time, p switches $S.p$ from *dominated* to *Dominator* if the neighbor it points out with $cur.p$ has a greater color (even if it is a *Dominator*).

We now show the correctness of Protocol $\mathcal{MIS}$ (Theorem 5). We then show in Theorem 6 that Protocol $\mathcal{MIS}$ is $\Diamond$-$(\lfloor \frac{\mathcal{L}_{max}+1}{2} \rfloor, 1)$-*stable* where $\mathcal{L}_{max}$ is the length (number of edges) of the longest elementary path in the network.

We show that Protocol $\mathcal{MIS}$ stabilizes to the *MIS predicate* in two steps: (1) We first show that any silent configuration of Protocol $\mathcal{MIS}$ satisfies the *MIS predicate*. (2) We then show that Protocol $\mathcal{MIS}$ reaches a silent configuration starting from any configuration in $O(\Delta \sharp \mathcal{C})$ rounds where $\sharp \mathcal{C}$ is the number of colors used in the network.

Lemma 3 *Any silent configuration of Protocol $\mathcal{MIS}$ satisfies the* MIS predicate.

Proof. The silent configurations of Protocol $\mathcal{MIS}$ are those from which all the S variables are fixed (remember that S is the only communication variable of Protocol $\mathcal{MIS}$).

So, in such a configuration γ, any *Dominator* (*i.e.* any process satisfying $S = Dominator$) has no neighbor that is also a *Dominator*, otherwise at least one of the *Dominator* process eventually becomes a *dominated* process (*i.e.*, a process satisfying $S = dominated$) following the first action of the protocol. Hence, the set of *Dominator* processes in γ is an independent set.

Moreover, any *dominated* process has a *Dominator* as neighbor in γ. Actually the neighbor pointed out by the *cur*-pointer is a *Dominator*. Hence the independent set in γ is maximal. □

Notation 1 *In the following, we use these notations:*

- $CSET = \{C.p, p \in \Pi\}$,
- $\sharp\mathcal{C} = |CSET|$, *and*
- $\forall c \in CSET$, $\mathcal{R}(c) = |\{c' \in CSET, c' \prec c\}|$.

Lemma 4 *Starting from any configuration, any computation of Protocol $\mathcal{MIS}$ reaches a silent configuration in at most $\Delta \times \sharp\mathcal{C}$ rounds.*

Proof. To show this lemma we prove the following induction: $\forall p \in \Pi, \mathcal{R}(C.p) = i$, the variable $S.p$ (the only communication variable) is fixed after at most $\Delta \times (i+1)$ rounds. The lemma will be then deduced by the fact that: $\forall c \in CSET$, $0 \leq \mathcal{R}(c) < \sharp\mathcal{C}$.

Case $i = 0$. Let p be a process such that $\mathcal{R}(p) = 0$ (such a process exists by Lemma 4). If $S.p = Dominator$ in the initial configuration, then $S.p$ remains equal to *Dominator* forever because $\forall q \in \Gamma.p, C.p \prec C.q$. If $S.p = dominated$ in the initial configuration, then the neighbor q pointed out by $cur.p$ satisfies $C.p \prec C.q$. So, p is enabled to switch $S.p$ to *Dominator*. Thus, p switches $S.p$ to *Dominator* in at most one round and then $S.p$ remains equal to *Dominator* forever (as in the previous case). Hence, any process p such that $\mathcal{R}(p) = 0$ satisfies $S.p = Dominator$ forever in at most one round and the induction holds for $i = 0$.

Induction assumption: Assume that there exists k, $0 \leq k < \sharp\mathcal{C} - 1$ such that for every process p such that $\mathcal{R}(p) = k$, the variable $S.p$ is fixed in at most $\Delta \times (k+1)$ rounds.

Case $i = k + 1$. Let p be a process such that $\mathcal{R}(p) = k + 1$. After $\Delta \times (k+1)$ rounds, the S-variable of any p's neighbor q such that $C.q \prec C.p$ is fixed by induction assumption. Consider then the two following cases:

- $\forall q \in \Gamma.p$, $C.q \prec C.p$, $S.q = dominated$. In this case, every p's neighbor j satisfies $C.p \prec C.j \vee S.j = dominated$ forever. As a consequence, either $S.p$ is already equal to *Dominator* or p switches $S.p$ to *Dominator* in the next round and then $S.p$ is fixed forever. Hence, $S.p$ is fixed to *Dominator* in at most $\Delta \times (k+1) + 1$ rounds and the induction holds in this case.
- $\exists q \in \Gamma.p$, $C.q \prec C.p$, $S.q = Dominator$. In this case, p increments $cur.p$ until pointing out a neighbor j such that $C.j \prec C.p \wedge S.j = Dominator$: p increments $cur.p$ at most $\delta.p - 1$ times. Once $cur.p$ points out a neighbor j such that $C.j \prec C.p \wedge S.j = Dominator$, $S.p$ is definitely set to *dominated* (because $S.j = Dominator$ forever). So, in at most Δ additional rounds, $S.p$

is definitely set to *dominated*. Hence, the S-variable of every process p such that $\mathcal{R}(p) = k + 1$ is fixed after at most $\Delta \times (k + 2)$ rounds and the induction holds for $i = k + 1$ in this case.

□

By Lemmas 3 and 4, Protocol $\mathcal{MIS}$ converges from any configuration to the *MIS predicate* in at most $(\Delta + 1)n + 2$ rounds. Moreover, $\mathcal{MIS}$ is 1-*efficient* because when a process p executes an action in $\mathcal{MIS}$, it only reads the S-variable and the C-constant of its neighbor pointed out by *cur.p*. Hence, follows:

Theorem 5 *Protocol $\mathcal{MIS}$ (Figure 8) is a* 1*-efficient protocol that stabilizes to the* MIS predicate *in any locally-identified network.*

The following theorem shows a lower bound on the number of processes that are eventually "1-stable" (*i.e.*, processes that eventually read the communication state of the same neighbor at each step). Figure 9 gives an example that matches the lower bound.

Theorem 6 *Protocol $\mathcal{MIS}$ (Figure 8) is $\Diamond$-$(\lfloor \frac{\mathcal{L}_{max}+1}{2} \rfloor, 1)$-stable where $\mathcal{L}_{max}$ is the length (number of edges) of the longest elementary path in the network.*

Proof. Let $\mathcal{L}_{max}$ be the length (number of edges) of the longest elementary path in the network. Once stabilized, at most $\lceil \frac{\mathcal{L}_{max}+1}{2} \rceil$ processes in this path are $Dominators$, otherwise at least two $Dominators$ are neighbors and the system is not stabilized. As a consequense, at least $\lfloor \frac{\mathcal{L}_{max}+1}{2} \rfloor$ processes are *dominated* in a silent configuration and Protocol $\mathcal{MIS}$ is $\Diamond$-$(\lfloor \frac{\mathcal{L}_{max}+1}{2} \rfloor, 1)$-*stable*. □

5.3 Maximal Matching

We now consider the *maximal matching* problem. A matching of the network is a subset of edges in which no pair of edges has a common incident process. A matching M is *maximal* if no proper superset of M is also a matching.

In *maximal matching problem*, each process p computes $\delta.p$ local Boolean functions $inMM[q].p$ (one for each neighbor q) that decide if the edge $\{p, q\}$ is in the maximal matching. The *maximal matching predicate* is *true* if and only if the subset of edges $\{\{p, q\} \in E, inMM[q].p \vee inMM[p].q\}$ is a *maximal matching* of the network. In any self-stabilizing maximal matching protocol, the legitimate configurations are those satisfying the *maximal matching predicate*.

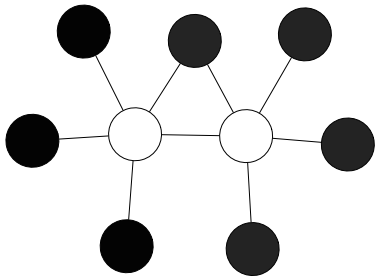

Figure 9: Example that matches the lower bound (the *Dominator* are the black nodes; the white nodes are the *dominated*)

We propose in Figure 10 a 1-*efficient* protocol that stabilizes to the *maximal matching* predicate. In the following we refer to this protocol as Protocol $\mathcal{MATCHING}$. The proposed protocol working in arbitrary networks still assuming the (local) coloring on processes.

Communication Variables:
$M.p \in \{true, false\}$

$PR.p \in \{0 \dots \delta.p\}$

Communication Constant:
$C.p$: color

Internal Variable:
$cur.p \in [1 \dots \delta.p]$

Predicate:
$PRmarried(p) \equiv (PR.p = cur.p \wedge PR.(cur.p) = p)$

Actions:

$(PR.p \notin \{0, cur.p\})$	$\rightarrow$	$PR.p \leftarrow cur.p$
$(M.p \neq PRmarried(p))$	$\rightarrow$	$M.p \leftarrow PRmarried(p)$
$(PR.p = 0 \wedge PR.(cur.p) = p)$	$\rightarrow$	$PR.p \leftarrow cur.p$
$(PR.p = cur.p \wedge PR.(cur.p) \neq p \wedge (M.(cur.p) \vee C.(cur.p) \prec C.p))$	$\rightarrow$	$PR.p \leftarrow 0$
$(PR.p = 0 \wedge PR.(cur.p) = 0 \wedge C.p \prec C.(cur.p) \wedge \neg M.(cur.p))$	$\rightarrow$	$PR.p \leftarrow cur.p$
$(PR.p = 0 \wedge (PR.(cur.p) \neq 0 \vee C.(cur.p) \prec C.p \vee M.(cur.p)))$	$\rightarrow$	$cur.p \leftarrow (cur.p \bmod \delta.p) + 1$

Figure 10: Protocol $\mathcal{MATCHING}$ For any process p

Protocol $\mathcal{MATCHING}$ derives from the protocol in [17], but with some adaptations to get the 1-*efficiency*. As previously, each process p has the communication constant color $C.p$ and uses the internal *cur*-pointer to designate the current neighbor from which it reads the communication variables.

The basic principle of the protocol is to create pairs of *married* neighboring processes, the edges linking such pairs being in the maximal matching. To that goal, every process p maintains the variable $PR.p$. Either $PR.p$ points out a neighbor or is equal to 0. Two neighboring processes are *married* if and only if their PR-values point out to each other. A process that is not married is said *unmarried*. The predicate $PRmarried(p)$ states if the process p is currently married, or not. Hence, for every process p and every p's neighbor q, $inMM[q].p \equiv (PRmarried(p) \wedge PR.p = q)$. If $PR.p = 0$, then this means that p is unmarried and does not currently try to get married. In this case, p is said *free*. If $PR.p \neq 0$, then p is either married or tries to get married with the neighbor pointed out by $PR.p$. Hence, the value of $PR.p$ is not sufficient to allow all neighbors of p to determine its current status (married or unmarried). We use the Boolean variable $M.p$ to let neighboring processes of p know if p is married or not.

Using these variables, the protocol is composed of six actions (ordered from the highest to the lowest priority). Using these actions, each process p applies the following strategies:

- p is only allowed to be (or try to get) married with the neighbor pointed out by $cur.p$. So, if $PR.p \notin \{0, cur.p\}$ then $PR.p$ is set to $cur.p$. Actually, if $PR.p = q$ such that $q \notin \{0, cur.p\}$, then $PR.p = q$ since the initial configuration.

- p must inform its neighbors of its current status, *i.e.* married or unmarried, using $M.p$. To compute the value of $M.p$ we use the predicate $PRmarried(p)$: if $M.p \neq PRmarried(p)$, then $M.p$ is set to $PRmarried(p)$.

- If p is free ($PR.p = 0$) and p is pointed out by the PR-variable of a neighbor q, then this means that q proposes to p to get married. In this case, p accepts by setting $PR.p$ to q (this rule allows to extend the matching).

- p resets $PR.p$ to 0 when the neighbor pointed out by $PR.p$ (i) is married with another process or (ii) has a lower color than p (*w.r.t.*, $\prec$). Condition (i) prevents p to wait for an already married process. Condition (ii) is used to break the initial cycles of PR-values.

- If p is free, then it must try to get married. The two last rules achieve this goal. p tries to find a neighbor that is free and having a higher color than itself (to prevent cycle creation). So, p increments $cur.p$ until finding an neighbor that matches this condition. In this latter case, p sets $PR.p$ to $cur.p$ in order to propose a marriage.

We now show the correctness of Protocol $\mathcal{MATCHING}$ (Theorem 7). We then show in Theorem 8 that Protocol $\mathcal{MATCHING}$ is $\Diamond$-$(\lceil \frac{2m}{2\Delta-1} \rceil, 1)$-*stable*.

Lemma 5 *In any silent configuration of Protocol $\mathcal{MATCHING}$, every process is either* free *or* married*.*

Proof. Assume, by the contradiction, that there is a silent configuration of $\mathcal{MATCHING}$ where there is a process p_0 that is neither *free* nor *married*. Then, by definition, $PR.p_0 = p_1$ such that $p_1 \neq 0$ (p_0 is not *free*) and $PR.p_1 \neq p_0$ (p_0 is *unmarried*). Also, $cur.p_0 = p_1$ otherwise p_0 is enabled to set $PR.p_0$ to $cur.p_0$, this contradicts the facts that the configuration is silent. Similarly, the fact that p is *unmarried* implies that $M.p_0 = false$.

As $PR.p_1 \neq p_0$ and $cur.p_0 = p_1$, we have $M.p_1 = false$ and $C.p_0 \prec C.p_1$ otherwise p_0 is enabled to set $PR.p_0$ to 0 and the configuration is not silent, a contradiction. In addition, $M.p_1 = false$ implies that p_1 is *unmarried*. Also, p_1 cannot be *free* otherwise p_1 eventually modify $PR.p_1$ (in the worst case, p_1 increments $cur.p_1$ until $cur.p_1 = p_0$ and then sets $PR.p_1$ to p_0). To sum up, p_1 is a neighbor of p_0 such that $C.p_0 \prec C.p_1$ and that is either *free* nor *married*.

Repeating the same argument for p_1 as we just did for p_0, it follows that p_1 has a neighbor p_2 such that $C.p_1 \prec C.p_2$ and that is either *free* nor *married*, and so on.

However, the sequence of processes p_0, p_1, $p_2 \ldots$ cannot be extended indefinitely since each process must have a lower color than its preceding one. Hence, this contradicts the initial assumption. □

Lemma 6 *Any silent configuration of Protocol $\mathcal{MATCHING}$ satisfies the* maximal matching predicate*.*

Proof. We show this lemma in two steps: (*i*) First we show that, in a silent configuration, the set A of edges $\{p, q\}$ such that $(PRmarried(p) \wedge PR.p = q)$ is a matching. (*ii*) Then, we show that this matching is a maximal.

(*i*) Consider any process p. By Lemma 5, in a silent configuration p is either *free* or *married*. If p is *free*, then p is incident of no edge in A. If p is *married*, then p is incident of exactly one edge in A, *i.e.*, the edge linking p and its neighbor pointed out by $PR.p$. Hence, in a silent configuration every process is incident of at most one edge in A, which proves that A is a matching.

(*ii*) Assume, by the contradiction, that there is a silent configuration γ where A is not maximal. Then, by Lemma 5, there is two neighbors p and q that are

free in γ. Following the two last actions of the protocol, at least one of them eventually modifies its PR-variable, this contradicts the fact that γ is silent.

□

Lemma 7 *After the first round, every process p satisfies $PR.p \in \{0, cur.p\}$ forever.*

Proof. Let p be a process.

First, if $PR.p \in \{0, cur.p\}$, then $PR.p \in \{0, cur.p\}$ forever because $PR.p$ can ony be set to 0 or $cur.p$ and $cur.p$ can be modified only if $PR.p = 0$.

Assume then that $PR.p \notin \{0, cur.p\}$. In this case, the first action (the one with the highest priority) of the protocol is enabled at p. So, $PR.p$ and $cur.p$ cannot be modified before execute this action: the action is continuously enabled. Hence, p sets $PR.p$ to $cur.p$ in at most one round and then $PR.p \in \{0, cur.p\}$ holds forever.

Hence, after the first round, every process p satisfies $PR.p \in \{0, cur.p\}$ forever. □

Lemma 8 *Let $A \in \Pi$ be a maximal connected subset of* unmarried *processes in some configuration after the first round. If $|A| \geq 2$, then after at most $2\Delta + 2$ rounds the size of A decreases by at least 2.*

Proof. Let S be the suffix of the computation that starts after the end of the first round. Let $\Gamma(A)$ be the set of process p such that $p \notin A$ and p has a neighbor $q \in A$.

First the size of A cannot increase because once *married*, a process remains *married* forever. Assume now, by the contradiction, that A does not decrease of at least 2 during $2\Delta+2$ rounds in S. This implies that no two process of A get married during this period.

Let S' be the prefix of S containing $2\Delta + 2$ rounds.

We first show that after one round in S', every process p satisfies: $(p \in \Gamma(A)) \Rightarrow (M.p = true) \wedge (p \in A) \Rightarrow (M.p = false)$. First, by definition of A, if $p \in \Gamma(A)$, p is *married*. So, if $M.p = true$ already holds at the beginning of the round, then it remains $true$ forever. Otherwise, p is continuously enabled to set $M.p$ to $true$ using the second action of the protocol and, as this action is the enabled action of p with the highest priority (by Lemma 7, the first action of p is disabled), p executes it in at most one round and then $M.p = true$ holds forever. If p is in A, then p is *unmarried* and, by contradiction assumption, p remains unmarried in S'. So, using a similar reasonning, in at most one round, $M.p = false$ holds in S'.

We now show that after two rounds in S', for every process p, if $PR.p \neq 0$, then $PR.p \neq 0$ holds until the end of S'. First, every process p that is enabled to set $PR.p$ to 0 (the fourth rule of the protocol) at the beginining of the third round in S' is continuously enabled because M and C are constant for every process and the neighbor q pointed out by $cur.p$ never set $PR.q$ to p (otherwise p becomes married, which contradicts the contradiction assumption). As the fourth action of p is the enabled action having the highest priority (remember that p must not execute its third action by contradiction assumption), p sets $PR.p$ to 0 in at most one round. Consider then the processes that sets $PR.p$ to 0 or that are disabled to execute the fourth action at the beginning of the round. These processes cannot be enabled again to execute the fourth action in S' because M and C are constant and a process p never points out using $PR.p$ to a process q such that $M.q \vee C.q \prec C.p$. Hence, after two rounds, for every process p, if $PR.p \neq 0$, then $PR.p \neq 0$ holds until the end of S'.

We now show that after $\Delta+2$ rounds in S', for every process p in A, we have either (1) $PR.p = q$, $q \in A$ or (2) $PR.p = 0$ and every neighbor $q \in A$ satisfies $PR.q \in A$. First, $PR.p = cur.p$ by Lemma 7. Then, as p is never married, $PR.(cur.p) \neq p$. Finally, from the previous case, after two round in S', no process can execute the fourth action. Hence, from the guard of the fourth action, we can deduce that $M.q = false$ and, as a consequence, $q \in A$ (see first part of the proof) which proves (1). We now show (2) by the contradiction. Assume then that after $\Delta + 2$ rounds there is two neighboring processes p and q in A such that $PR.p = 0$ and $PR.q = 0$. After 2 rounds in S', only the two last actions of the protocol can be executed in S' by p or q. Then, p and q executes the last action of the protocol at each round (as they are assumed to satisfy $PR = 0$ after $\Delta + 2$ rounds, they cannot execute the fifth action). Now, both p and q satisfy $M = false$ and $PR = 0$. Also, either $C.p \prec C.q$ or $C.q \prec C.p$. So, both p and q cannot execute the last action of the protocol Δ times, a contradiction.

We now show that in at most $2\Delta + 2$ rounds, at least two neighboring processes in A get married. First, after $\Delta + 2$ rounds in S', there is some process $p \in A$ such that $PR.p = 0$ and there exists a neighbor q such that $PR.q = p$ otherwise there at least one process that is enabled to set its PR-variable to 0 (to break the PR-cycle), which contradicts the second part of the proof. As previously, p executes at least one of the two last actions of the protocol at each round. So, in at most $\Delta - 1$ rounds, p points out using $cur.p$ a process such that $PR.(cur.p) = p$ and then get married with the process pointed out by $cur.p$ in at most one additionnal round, which contradicts the contradiction assumption.

Hence, before the end of S' at least two processes get married, which proves the lemma. □

Lemma 9 *Starting from any configuration, any computation of Protocol $\mathcal{MATCHING}$ reaches a silent configuration in at most $(\Delta + 1)n + 2$ rounds.*

Proof. First, the number of *married* processes cannot decrease. Then, after the first round and until there is a maximal matching in the system, the number of *married* processes increases by at least 2 every $2\Delta + 2$ rounds by Lemma 8. Hence, there is a maximal matching into the networks after at most $(\Delta + 1)n + 1$ rounds. Once maximal matching is available in the network, one more round is necessary so that every *married* process p satisfies $M.p = true$ and every *ummarried* process p satisfies $PR.p = 0$. Hence, starting from any initial configuration, the system reaches a silent configuration in at most $(\Delta + 1)n + 2$ rounds. □

By Lemmas 6 and 9, Protocol $\mathcal{MATCHING}$ converges from any configuration to the *maximal matching predicate*. Moreover, $\mathcal{MATCHING}$ is 1-*efficient* because when a process p executes an action in Protocol $\mathcal{MATCHING}$, it only reads the communication variables of its neighbor pointed out by $cur.p$. Hence, follows:

Theorem 7 *Protocol $\mathcal{MATCHING}$ (Figure 10) is a* 1*-efficient protocol that stabilizes to the* maximal matching predicate *in any locally-identified network.*

The following theorem shows a lower bound on the number of processes that are eventually "1-stable" (*i.e.*, processes that eventually read the communication state of the same neighbor at each step). Figure 11 gives an example that matches the lower bound.

Theorem 8 *Protocol $\mathcal{MATCHING}$ (Figure 10) is $\Diamond$-$(2\lceil\frac{m}{2\Delta-1}\rceil, 1)$-stable where m.*

Proof. From [6], we know that any maximal matching in a graph has a size at least $\lceil\frac{m}{2\Delta-1}\rceil$ edges. So, as a process belongs to at most one matched edge, we can conclude that at least $2\lceil\frac{m}{2\Delta-1}\rceil$ processes are eventually matched. As a consequence, Protocol $\mathcal{MATCHING}$ is $\Diamond$-$(2\lceil\frac{m}{2\Delta-1}\rceil, 1)$-*stable*. □

6 Concluding remarks

We focused on improving communication efficiency of self-stabilizing protocols that eventually reach a global fixed point, and devised how much gain can be expected

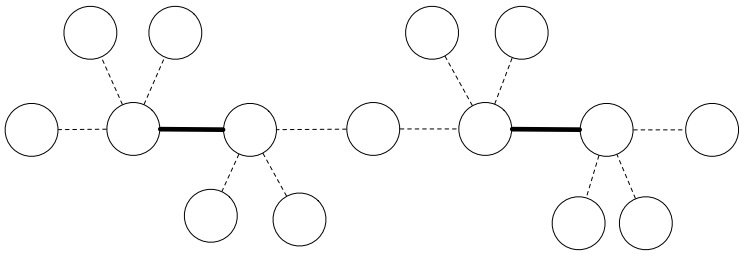

Figure 11: Example that matches the lower bound (the matched edges are in bold): $\Delta = 4$ and $m = 14$

when implementing those protocols in a realistic model. Our results demonstrate the task difficulty, as most systematic improvements are impossible to get, yet also shows that some global improvement can be achieved over the least-overhead solutions known so far, the so-called *local checking* self-stabilizing protocols.

While we demonstrated the effectiveness of our scheme to reduce communication need on several local checking examples, the possibility of designing an efficient general transformer for protocols matching the local checking paradigm remains an open question. This transformer would allow to easily get more efficient communication in the stabilized phase or in absence of faults, but the effectiveness of the transformed protocol in the stabilizing phase is yet to be known.

References

[1] Marcos Kawazoe Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, and Sam Toueg. On implementing omega with weak reliability and synchrony assumptions. In *PODC*, pages 306–314, 2003.

[2] Marcos Kawazoe Aguilera, Carole Delporte-Gallet, Hugues Fauconnier, and Sam Toueg. Communication-efficient leader election and consensus with limited link synchrony. In *PODC*, pages 328–337, 2004.

[3] Baruch Awerbuch, Boaz Patt-Shamir, and George Varghese. Self-stabilization by local checking and correction (extended abstract). In *FOCS*, pages 268–277. IEEE, 1991.

[4] Baruch Awerbuch, Boaz Patt-Shamir, George Varghese, and Shlomi Dolev. Self-stabilization by local checking and global reset (extended abstract). In Gerard

Tel and Paul M. B. Vitányi, editors, *Distributed Algorithms, 8th International Workshop, WDAG '94*, volume 857 of *Lecture Notes in Computer Science*, pages 326–339. Springer, 1994.

[5] Joffroy Beauquier, Sylvie Delaët, Shlomi Dolev, and Sébastien Tixeuil. Transient fault detectors. *Distributed Computing*, 20(1):39–51, 2007.

[6] Therese C. Biedl, Erik D. Demaine, Christian A. Duncan, Rudolf Fleischer, and Stephen G. Kobourov. Tight bounds on maximal and maximum matchings. *Discrete Mathematics*, 285(1-3):7–15, 2004.

[7] Carole Delporte-Gallet, Stéphane Devismes, and Hugues Fauconnier. Robust stabilizing leader election. In Toshimitsu Masuzawa and Sébastien Tixeuil, editors, *SSS*, volume 4838 of *Lecture Notes in Computer Science*, pages 219–233. Springer, 2007.

[8] Edsger W. Dijkstra. Self-stabilizing systems in spite of distributed control. *Commun. ACM*, 17(11):643–644, 1974.

[9] S. Dolev. *Self-stabilization*. MIT Press, March 2000.

[10] Shlomi Dolev, Mohamed G. Gouda, and Marco Schneider. Memory requirements for silent stabilization. *Acta Inf.*, 36(6):447–462, 1999.

[11] Shlomi Dolev, Amos Israeli, and Shlomo Moran. Resource bounds for self-stabilizing message-driven protocols. *SIAM J. Comput.*, 26(1):273–290, 1997.

[12] Maria Gradinariu and Sébastien Tixeuil. Self-stabilizing vertex coloring of arbitrary graphs. In *International Conference on Principles of Distributed Systems (OPODIS'2000)*, pages 55–70, Paris, France, December 2000.

[13] Michiyo Ikeda, Sayaka Kamei, and Hirotsugu Kakugawa. A space-optimal self-stabilizing algorithm for the maximal independent set problem. In *the Third International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT)*, pages 70–74, Kanazawa, Japan, September 2002.

[14] Amos Israeli and Marc Jalfon. Token management schemes and random walks yield self-stabilizing mutual exclusion. In *Proceedings of the Ninth Annual ACM Symposium on Principles of Distributed Computing*, pages 119–131, 1990.

[15] Shmuel Katz and Kenneth J. Perry. Self-stabilizing extensions for message-passing systems. *Distributed Computing*, 7(1):17–26, 1993.

[16] Mikel Larrea, Antonio Fernández, and Sergio Arévalo. Optimal implementation of the weakest failure detector for solving consensus. In *SRDS*, pages 52–59, 2000.

[17] Fredrik Manne, Morten Mjelde, Laurence Pilard, and Sébastien Tixeuil. A new self-stabilizing maximal matching algorithm. In *Proceedings of the 14th International Colloquium on Structural Information and Communication Complexity (Sirocco 2007)*, volume 4474, pages 96–108. Springer Verlag, June 2007.

Unité de recherche INRIA Futurs
Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)

Unité de recherche INRIA Lorraine : LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399